QEMU as prototyping platform for seL4 systems

Axel Heider, HENSOLDT Cyber

seL4 summit, 2022-10-10

HENSOLDT
Detect and Protect.

Why QEMU?

- Works well for things running above the hardware abstraction layer
- Simplifies cross-platform development
- Reasonably deterministic or fast (choose one)
- Scales nicely for CI pipelines
- Available to everybody
- No hardware instrumentation needed, no “hick-ups“
- No debug/trace hardware needed

Things to keep in mind

- Works at instruction level (or “translation block” level) only
  - not cycle accurate, no simulation for pipeline
  - no caches, no write buffer

- Simplified Hardware simulation
  - registers might be dummies
  - no FIFOs, no accurate I/O timing

- Documentation could be better
  - FOSDEM2018: Finding your way through the QEMU parameter jungle
  - Xilinx QEMU fork

- Version Numbering
  - Release every 4 months (April, August, December)
  - 2018-08 is v3.0 (not v2.13), since v4.0 (2019-04) a major release every year

Usage of QEMU in seL4 CI

- seL4test
  - ia32/x86_64
    - PC99 (Nehalem)
  - ARMv7
    - SABRE (sabrelite)
    - ZYNQ7000 (xilinx-zynq-a9)
  - ARMv8
    - ARMVIRT (virt)
  - RISC-V
    - SPIKE32 (build for “spike”, running on “virt”)
    - SPIKE64 (spike)

- camkes-vm
  - ARMv8 (virt) for vm_minimal example

seL4test on QEMU

- Cache tests are disabled, fail because there is no cache
- One failing scheduler test disabled, seems a test implementation issue
- Timer tests disabled
  - “sabrelite”: QEMU mainline still misses EPIT timer fix
  - “xilinx-zynq-a9”: unstable? Seems to work in QEMU v7.1
  - “virt” has no timer peripheral (the RTC can’t be used)

- Other working platforms
  - ARMv7 “virt“ (no timer)
  - ARMv8 “xlnx-zcu102“ (timer test fail due to frequency settings)

- Dead simulation platforms
  - „raspi3“: seL4 does not boot. Anybody?

Which QEMU to use?

- Whatever works best for what you actually want...
- For TRENTOS CI:
  - „sabrelite“
    - QEMU with EPIT fix
    - native drivers for NIC and SD-Card
  - „xilinx-zynq-a9“
    - adding native NIC support still on ToDo list
    - Simulate NICs via TRENTOS “ChanMux” → UART → TestFramework → TAPs
  - „*virt-sel4*“
    - … work in progress as unified solution for ARM and RISC-V

QEMU to simulate our MiG-V SoC

- Customization
  - Started from RISC-V “spike” platform code base
    - Adapt memory configuration
      - 2 RAM areas, 1 ROM area, 1 Flash area
      - trap writes to ROM area, init via image
  - Rebased to sifive board emulation
    - PLIC support
    - Replace spike’s HTIF console by a “real” UART
    - Add UARTs for I/O channel, add timer peripherals
- allows MiG-V specific development without FPGA/Board access
    - Bootloader/SBI/Loader
    - ROM version of seL4
    - Tooling/Workflow for system deployment

QEMU virt platform (RISC-V, ARM)

- Why stick to a board emulation actually?

- Configure via “`-machine virt[,…],dumpdtb=<fielname> -cpu <name> ...`“
  - ARM: GICv2/3/4, SMMUv3, Virtualization, TrustZone …
  - RISC-V: (A)PLIC, (A)CLINT ….
  - See “`-machine virt,help`“ and “`-cpu help`” or details

- seL4 build workflow
  - Invoke seL4 build system with seL4 config params
  - Build QEMU config and extract device tree
  - Build seL4 system against with that device tree
  - Use „simulate“ script to run seL4 system on QEMU with this configuration

QEMU “virt“ pitfalls

- Fimware dependencies
  - aarch64/virt needs "`efi-virtio.rom`"
    - package “ipxe-qemu“ is not enforced for “qemu-system-arm“
    - Use dummy file, or “-nic none“?
  - riscv/virt wants "`opensbi-riscv64-generic-fw_dynamic.bin`"
    - use „--bios `<seL4 image>`“ in QEMU v5.x and higher
    - (fix search paths)
- Hard-coded assumptions in seL4:
  - VMM: drop “GIC_IRQ_PHANDLE“ and parse DTS instead
  - Boot: Check passed DTB matches device tree used when building

Custom QEMU with “virt-sel4“

- Extend „virt“ to have a generic, seL4-friendly development platform
  - Motivation: run all VM examples on “arm-virt”
- Add peripherals:
  - timers for userland (SP804 on ARM)
  - serial ports for simple I/O channels (PL011 on ARM)
- Add a seL4 aware tracepoint backend
  - replace the “debug log trace”
  - Inspired by “My other machine is virtual” (Linaro Connect YVR18-118)
- Add QEMU binaries to existing docker container
  - Upstreaming to extend “virt” seems unlikely

What's next?

- Consider other emulators – Renode?

Axel Heider

axel.heider@hensoldt.net

HENSOLDT Cyber GmbH
Willy-Messerschmitt-Straße 3
82024 Taufkirchen
www.hensoldt-cyber.com

QEMU built-in tracing

- `-d <option,option...> -D <logfile>`
  - `in_asm` show assembly (one for each compiled TB, "`-singelstep`")

    No longer works in v7.1, just says "`OBJD-T: 73c23f91`"
    requires building QEMU with libcapstone support to see "`add x19, x19, #0xff0`"

  - `int` show interrupts/exceptions
  - `exec` show each executed TB (and the CPU ID)

    `Trace <`**`CPU-ID`**`>: <tb> [<tb->tc.ptr>/<`**`pc`**`>/<tb-flags>/<tb-flags>] <symbol>`

  - `nochain` don't chain compiled TBs
  - `tid` **new in v7.1**, separate logs per CPU (use „-D `logfile-%d`")

  - `cpu` show CPU registers before entering a TB
  - unimp log unimplemented functionality
  - `guest_errors` log invalid operations

- `-dfilter <range>[,<range>...]`:
  - Log for a certain range only
  - "`<start>`…`<end>`", "`<start>+<size>`", "`<start>-<size>`"

Making QEMU more deterministic

- “`-icount shift=N`“
  - CPU executes one TB every 2^N ns of virtual time.
  - Give a deterministic (virtual) timer
  - Implicitly disables MTTCG

- SMP
  - MT-TCG (Multi Thread Tiny Code Generator) since V2.9 (2017)
  - CPUs runs as separate threads
  - Disable with “`--accel tcg,thread=single`“
    - Back to Round robin, one TB at a time
    - Need “`-singelstep`“ for single-instruction TBs

Getting things into QEMU

- Via “`--kernel <elf>`”
  - also load symbols, shown in traces
  - Change in v5.1 for RISC-V
    - “`--bios <elf>`” for M-Mode
    - “`--kernel`“ start in S-Mode with bundled OpenSBI firmware image

      ...or complains "`opensbi-riscv64-generic-fw.bin`" is missing

- “`--device loader,…`“
  - load binary or ELF (with symbols):

    ```
    ”...file=<file>[,addr=<addr>][,force-raw=<raw>]“
    ```

  - Set Memory:

    ```
    ”...addr=<addr>,data=<data>,data-len=<data-len>
          [,data-be=true]“
    ```